07F U8974

53

XP-002238913

P.D. 26-09-1999
P. 1-13 (13)

Applying Visual Basic for Human Machine Interface Applications

John B. Weber
Vice President, Technology
Software Toolbox
Charlotte, NC 28273

KEYWORDS

Visual Basic, VB, HMI, Human Machine Interface, Object Based Software, Objects, Connectivity, COM, DCOM, OPC, ActiveX, PLC Connectivity, Plant-Floor Connectivity.

ABSTRACT

Object based technologies and the Visual Basic programming language have gained widespread acceptance in the general marketplace with over 3,000,000 users worldwide. This paper will explore the technical issues that must be addressed in providing information delivery from plant floor devices to the enterprise in human machine interface applications utilizing Visual Basic technologies. Issues to be addressed will include PLC and control system connectivity, database connectivity, recipe download, logging, trending, and alarming. The paper will explore how off-the-shelf technologies based on specifications such as COM, ActiveX and OPC can be used with applications developed in Visual Basic to address these issues and provide flexible, cost-effective information delivery systems. The audience for this paper is the control systems engineer with little or no experience with Visual Basic but who is familiar with PLCs, is comfortable using a PC running the Windows operating system, and may have created HMI applications in the past using other tools.

INTRODUCTION

The primary goal of a human-machine interface (HMI) is to assist the operator in running a machine and managing a process. A good HMI will increase the productivity of the operator and machine, increase uptime, and assist in providing consistent product quality. The required functionality of an HMI will vary based upon the type and complexity of product produced, the type of machinery used, the skills of the operator, and the degree of automation of the machinery. The types of functionality typically included are:

Functionality	Purpose
Graphic Displays	To provide information about machine operation and status to the operator in a format that allows for easy interpretation and determination of need for action
User Input	To facilitate inputs from the operator to adjust machine operation, perform machine setups, and respond to events
Data Logging & Storage	To provide for the storage of historical machine operating data for part traceability and analysis of ways to improve quality, productivity, and uptime. Also used to store and retrieve machine setup data where

	needed.
Trending	To provide a means for visual analysis of data on current or past machine operation
Alarming	To provide notification to the operator of abnormal operating conditions and events.

The available approaches for providing HMI functionality typically include:

- Hardwired interface devices - pilot lights, numeric displays, pushbuttons, and switches
- Proprietary hardware displays - terminals ranging in size from small 2 x 20 character displays with keypads to color flat panel touchcreens to full 15" color CRTs with sealed keyboards. These terminals typically run a proprietary operating system and are configured using software provided by the hardware manufacturer.
- Personal Computer (PC) based displays - these HMIs consist of an industrially hardened PC, display, and keyboard running a standard commercially available operating system and off-the-shelf software rather than software provided by the hardware manufacturer. The PCs may be networked to other HMIs in the plant to provide plant wide display functionality. The user, OEM, or system integrator configures the off-the-shelf software for the specific machine or process application using tools provided by the software vendor.

This paper focuses on the 3rd type of HMI option, the PC based HMI. Specifically, this paper focuses on the use of off-the-shelf technologies used with the Visual Basic development environment for creating HMI applications on the Windows 9x and NT operating systems. The technologies and tools used are described in the context of several sample applications showing the code used to create simple HMIs in Visual Basic.

OBJECT TECHNOLOGIES

Visual Basic[1] (VB) is a development and scripting language used on the Windows 9x and NT operating systems (Win32 operating systems) for the creation of application software. The language is designed for a broader audience with less formal programming experience and training than lower level languages such as C or C++. As the name implies, creating of applications is done in a visual click-and-drop environment with the focus on rapid development of user interfaces.

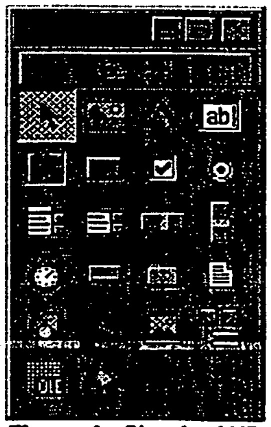


Figure 1 - Standard VB Toolbar

The user may select from a pallette of tools (Figure 1) that provide standard functionality such as command buttons, text boxes, labels, list boxes, dialog boxes, database connectivity, images, and scroll bars. These tools are known as objects. The user lays these objects out on the screen using a mouse to build simple operator interface screens.(Figure 2) The VB environment provides for

[1] Visual Basic, VB, COM, DCOM, ActiveX, Windows95, WindowsNT, and Win32 are acknowledged trademarks of the Microsoft Corporation. The author is not an employee of nor bears any official affiliation with Microsoft The names of these technologies are utilized not for commercial reasons but because they are terms used to refer to technologies that have become a part of the common language used when discussing software development for 80% of the desktop applications in use in the world today.

the automatic generation of events based on operator interaction with elements of the user interface. For example, if a user clicks on a command button the developer named `cmdPump1Start`, an event named `CmdPump1Start_Click` is automatically generated by Visual Basic. The developer writes one line of Visual Basic to place text into a text box named `txtPump1Status` when the command button is clicked: `txtPump1Status = "Pump 1 Started"`. The developer creates a compiled EXE of their application with one mouse click. With current technologies, the VB compiler generates native code for the Win32 operating systems instead of interpreted code, providing for performance that in many applications used to require a lower level language such as C++.

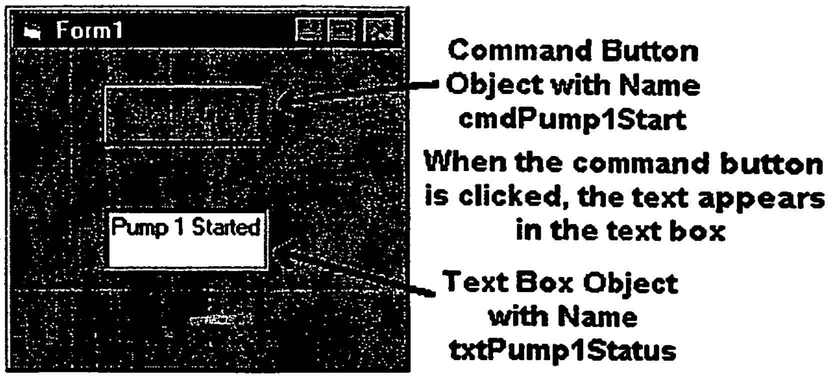
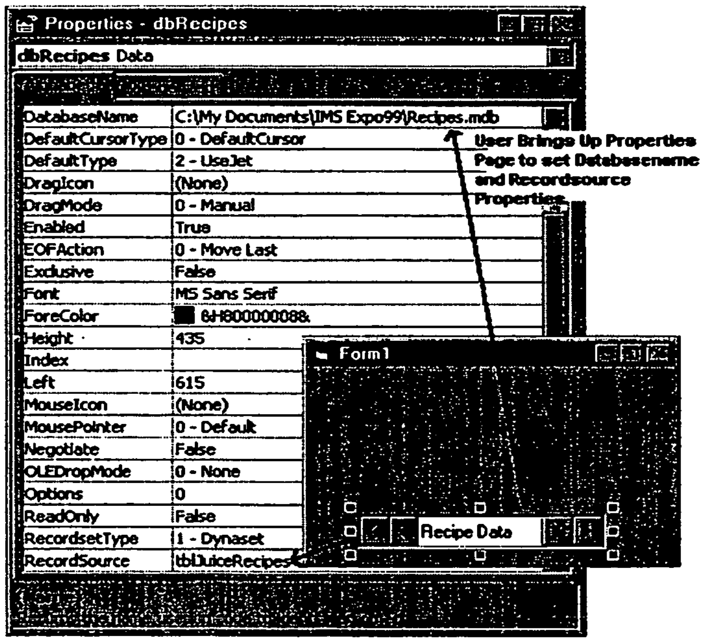

Form1
Pump 1 Started
Command Button Object with Name cmdPump1Start
When the command button is clicked, the text appears in the text box
Text Box Object with Name txtPump1Status

Figure 2 - Simple VB Application

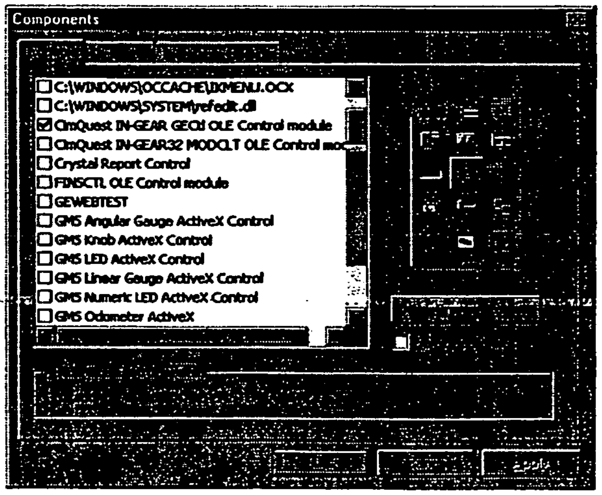
Connectivity to databases is accomplished using off-the-shelf tools as well. A data object on the toolbar can be added to a display. The user specifies a database file name in the `DatabaseName` property and table or query name in the `RecordSource` property in the data control as shown in Figure 3. Display of data is accomplished by simply setting the properties of a text box to connect its `DataSource` property to a field in the database setup on the data control. To scroll through records in the database, the user simply clicks on the provided < or > buttons on the screen.


Properties - dbRecipes
dbRecipes Data
User Brings Up Properties Page to set Databasename and Recordsource Properties
Form1
Recipe Data

DatabaseName	C:\My Documents\IMS Expo99\Recipes.mdb
DefaultCursorType	0 - DefaultCursor
DefaultType	2 - UseJet
DragIcon	(None)
DragMode	0 - Manual
Enabled	True
EOFAction	0 - Move Last
Exclusive	False
Font	MS Sans Serif
ForeColor	■ &H80000008&
Height	435
Index	
Left	615
MouseIcon	(None)
MousePointer	0 - Default
Negotiate	False
OLEDropMode	0 - None
Options	0
ReadOnly	False
RecordsetType	1 - Dynaset
RecordSource	tblJuiceRecipes

Figure 3 - Database Connectivity Using Data Control

Extensibility of the VB development environment is provided through the use of plug-in tools that are based on the Component Object Model or COM. All Win32 operating systems and applications are based on COM. These plug-in tools are objects that are loaded on the PC and added to the development environment through a simple point-and-click interface. (Figure 4).



☐ C:\WINDOWS\OCCACHE\IXMENU.OCX
☐ C:\WINDOWS\SYSTEM\refedit.dll
☑ CimQuest IN-GEAR GEOI OLE Control module
☐ CimQuest IN-GEAR32 MODCLT OLE Control mod
☐ Crystal Report Control
☐ FINSCTL OLE Control module
☐ GEWEBTEST
☐ GMS Angular Gauge ActiveX Control
☐ GMS Knob ActiveX Control
☐ GMS LED ActiveX Control
☐ GMS Linear Gauge ActiveX Control
☐ GMS Numeric LED ActiveX Control
☐ GMS Odometer ActiveX

Figure 4 - Adding Components and Tools to the VB Design Environment

Because of its simplified approach to application development with the focus on rapid development of user interfaces, database connectivity, and easy extensibility, Visual Basic has become a widely adopted technology in the broad market for software for PCs running Win32 operating systems. Over 3,000,000 users worldwide utilize Visual Basic to create their applications[2].

The industrial user is typically an engineer skilled in control systems design, PLC programming, and process control. With newer generations of engineers coming into plants with experience working with PCs going back into their primary schooling years, many of these engineers are more likely to be skilled at working with or creating software for use on personal computers. Nevertheless, these engineers have limited time and budget resources. When called upon to create an HMI application, they evaluate their options and seek the solution best suited to the needs of the application and their budgetary constraints.

Many engineers are investigating the use of Visual Basic for the creation of their HMI applications for several reasons. Many HMI applications only require a few user input screens and simple data logging and trending. In the cases of these applications, there are times when off-the-shelf HMI solutions provide more functionality at a higher cost than budgets allow for. Until recently, the use of VB for industrial HMI applications was limited by the performance of interpreted code at runtime and the lack of tools for industrial applications such as control system hardware connectivity and data visualization. New technologies based on COM have removed these barriers by extending the suite of tools and objects available to the Visual Basic developer. With these barriers gone, engineers can utilize Visual Basic and quickly create simple HMI applications. From a budgetary standpoint, the potential savings are large. Developers creating applications in Visual Basic can distribute their compiled EXE applications at minimal cost. No per machine runtime licenses are required by the provider of the development environment for the actual Visual Basic code the developer creates or the built-in objects (text boxes, command buttons, etc) they may use. Some third party plug-ins may require nominal per machine fees or none at all. From a long-term maintenance point-of-view, the fact that over 3,000,000 people worldwide know the Visual Basic language provides comfort that one will always be able to find someone with the aptitude and skills to understand the application source code and maintain it.

BUILDING A SIMPLE HMI AND CONNECTING TO A PLC

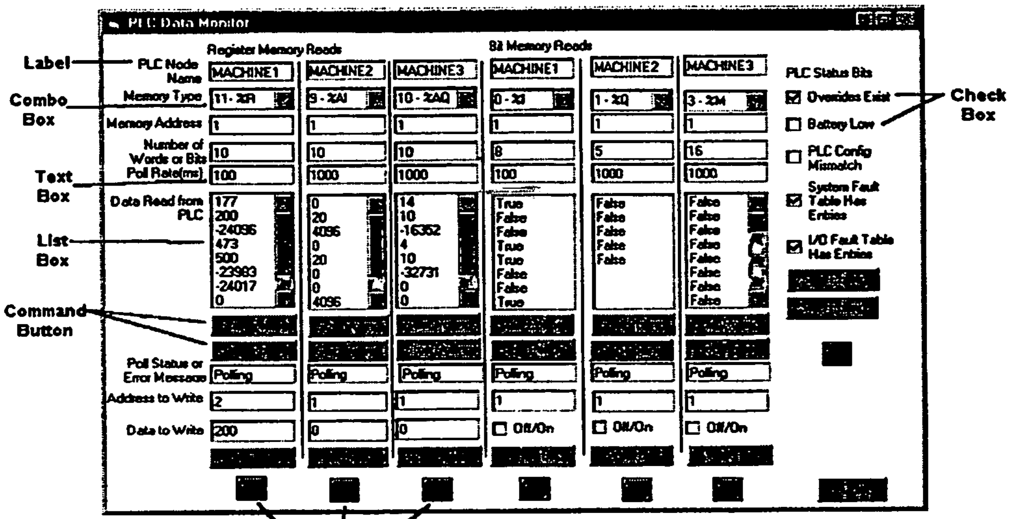
For ease of understanding in this paper, all Visual Basic code keywords, object names, properties, methods, and event names are written in `courier typeface`. Full code listings are done in this typeface.[3]

In the simplest form, an HMI will provide the ability to read data from a programmable logic controller (PLC) and write data back to the PLC. The display of data, input of data, and initiation of the read/write transactions from the user interface is handled using built in objects in Visual Basic: text boxes, labels, lists, and command buttons. The missing link is the connection to the PLC.

Because COM allows the user to add objects to Visual Basic for specific tasks, a range of tools has emerged in the general marketplace for providing connectivity. The objects that can be added to VB are

[2] "No Oxymoron with VBA", Neil Charney and Mike Gilbert in an interview with Software Strategies, January 1998, p.44.
[3] All code listings are provided for illustrative purposes only. Neither the author nor the ISA provides is liable for any use of this sample code in a production environment.

called ActiveX controls, formerly known as OCX's or OLE custom controls. These objects all have properties the developer sets to define the exact functionality the object will perform and methods the developer invokes to make the control perform its functions. In an ActiveX control used to communicate with a PLC, properties are set to specify the PLC node address, memory address, amount of data, etc. and methods are called to initiate read/write transactions with the PLCs. The objects handle all of the low level protocol formatting, hardware interfacing, and error checking needed to communicate to the PLC. The object generates events like the command button did when clicked in the first example to notify the application that data has been received from or written to the PLC.



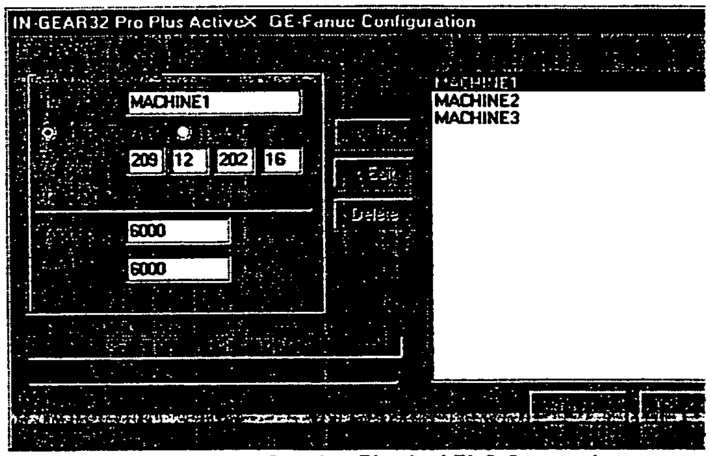
	Register Memory Reads			Bit Memory Reads		
PLC Node Name	MACHINE1	MACHINE2	MACHINE3	MACHINE1	MACHINE2	MACHINE3
Memory Type	11 - %R	9 - %AI	10 - %AQ	0 - %I	1 - %Q	3 - %M
Memory Address	1	1	1	1	1	1
Number of Words or Bits	10	10	10	8	5	16
Poll Rate(ms)	100	1000	1000	100	1000	1000
Data Read from PLC	177 200 -24096 473 500 -23983 -24017 0	0 20 4096 0 20 0 0 4096	14 10 -16352 4 10 -32731 0 0	True False False True True False False True	False False False False False	False False False False False False False False
Poll Status or Error Message	Polling	Polling	Polling	Polling	Polling	Polling
Address to Write	2	1	1	1	1	1
Data to Write	200	0	0	☐ Off/On	☐ Off/On	☐ Off/On

PLC Status Bits
☑ Overrides Exist
☐ Battery Low
☐ PLC Config Mismatch
☑ System Fault Table Has Entries
☑ I/O Fault Table Has Entries

Figure 5 - Simple HMI with PLC Reading and Writing Functionality

The example screen in Figure 5 shows a simple HMI built to perform simple PLC read/write functionality and points out the basic VB objects used for the user interface. The controls used to talk to the PLC are also shown. This paper will focus on the code required to perform the PLC connectivity function rather than the building of this form.

The PLC connectivity driver is an off-the-shelf ActiveX object that has been assigned the object name `PLCNode1`. In this case, the connection to the PLC will be made via TCP/IP Ethernet. First the user will use a built-in utility provided with the PLC communications ActiveX control to assign a meaningful name to each PLC that will be mapped to the IP address of the PLC. (Figure 6). Then the code shown in Listing 1 is entered in the Visual Basic editor. This code gets the user input data for the PLC node name, memory type to read, memory address, number of words, and poll rate from the text boxes and combo boxes on the form and maps it into the appropriate properties in the `PLCNode1` ActiveX control. Then it invokes the `.AutoPoll` method and passes the poll rate parameter in


IN-GEAR32 Pro Plus ActiveX GE-Fanuc Configuration

MACHINE1
209 12 202 16
6000
6000

MACHINE1
MACHINE2
MACHINE3

Figure 6 - Utility for Configuring Physical PLC Connections

Listing 1 - Code entered into the VB generated cmdPLCStartPollingReg1_Click event handlei

```
PLCNode1.Function = 0 'Set the function to read
PLCNode1.Host = RegPLCNodeAddress(0) 'get the PLC node name from the text box
PLCNode1.Address = RegAddress(0) 'Get the PLC register address to read
PLCNode1.Size = RegNumbWords(0) 'Get the number of words to read
PLCNode1.RegType = cmbRegReadType(0).ListIndex 'get the PLC memory type
PLCNode1.AutoPoll (RegPollRate(0)) 'start the polling of the PLC with the user supplied rate
txtNode1Status = "Polling"     'Update the display box to tell the user polling has begun
```

milliseconds obtained from the user input. This is the only code required to initiate the reads on a regular polled basis.

Listing 2 - Code entered into the PLCNode1_OnReadDone Event

```
RegReadData(0).Clear 'clear the display listbox
For i = 0 To PLCNode1.Size - 1 'loop through all points read
  'display the data on the form in the list box
  RegReadData(0).AddItem PLCNode1.WordVal(i)
Next i
```

When the PLCNode1 control gets data back from the PLC, the PLCNode1_OnReadDone event is automatically triggered by Visual Basic. The code shown in Listing 2 gets the number of words of data requested by the user and displays them in a list box.

Listing 3 - Code to pause reading, write a datapoint, and then restart the reading of data.

```
Private Sub cmdRegWrite_Click(Index As Integer)
    PLCNode1.AutoPoll (0) 'pause the reading
    PLCNode1.Size = 1    'Set size to one point
    PLCNode1.Address = RegAddressToWrite(0) 'get value to write
    'put value to write into the PLCNode1 ActiveX Control's data buffer
    PLCNode1.WordVal(0) = Int(RegValuetoWrite(0))
    PLCNode1.Function = 1 'set the PLCNode1 Control to Write
    PLCNode1.Trigger    'Trigger the control to write the data
End Sub

Private Sub PLCNode1_OnWriteDone()
    PLCNode1.Function = 0
    PLCNode1.Size = RegNumbWords(0)
    PLCNode1.Address = RegAddress(0)
    PLCNode1.AutoPoll (RegPollRate(0))
End Sub
```

To write data to the PLC requires the same actions but in reverse. The user inputs the memory address to write and the value to write in the text boxes on the user interface and clicks on the "Write Value" command button. The code shown in Listing 3 retrieves the address and data, stops the read polling, sets the ActiveX control's `.Function` property to write (value 1), puts the data into the control's data array in the first position with the `Wordval(0)` method and invokes the `.Trigger` method to write the data one time. When the `PLCNode1_OnWriteDone` event fires, the code sets the `.Function` propery back to read and invokes the `AutoPoll` method again to continue the reading of the PLC. Note that this is not the only method to provide this functionality. Reading can continue uninterrupted if the user is willing to place an extra instance of the PLC ActiveX control on the form and use it for writing, leaving the instance used for reading undisturbed.

In reviewing these code samples, note that for many of the user input objects, the names used are intuitive names. In Visual Basic the user has complete control over the object names. Also note that many of the object names include a subscript in parenthesis like an array - i.e. `RegReadData(0)`. These are called control arrays. The control array is a grouping of objects in a Visual Basic program having the same name when they are physically different objects on the screen. The different objects are distinguished from one another by their index shown in the parenthesis, i.e. 0 on `RegReadData(0)`. By utilizing control arrays, the user only has to remember one common name and then index through all instances of that object using the index number in parentheses. The index of the object can be set programmatically also, allowing the user to create looping structures that affect a number of controls as the code iterates through the object index in the control array.

The remainder of this sample HMI form was animated using the same code as shown in Listings 1 to 3 but with adjustments in the object names to reflect the different instances of the user input objects and

Downloading the data to the PLC is accomplished the same way data was written to the PLC in the simple read/write example. The `.Wordval` method is used to insert the data values to write into the PLC control `PLCMachine1` and the `.Trigger` method is invoked to write the data. When the `PLCMachine1_OnWriteDone` event fires, a message box is displayed to notify the user of the success.

Listing 4 - Code used to advance from one recipe record to another

```
Private Sub cmdNextRecipe_Click()
    If Not dbRecipes.Recordset.EOF Then
        dbRecipes.Recordset.MoveNext
    Else
        response = MsgBox("You have reached the end of the
        recipe list", vbOKOnly, "Advisory")
    End If
End Sub

Private Sub cmdPreviousRecipe_Click()
    If Not dbRecipes.Recordset.BOF Then
      dbRecipes.Recordset.MovePrevious
    Else
        response = MsgBox("You have reached the beginning of
        the recipe list", vbOKOnly, "Advisory")
    End If
End Sub
```

Obviously a full blown recipe management system would require the ability to edit the recipes, add recipes, upload recipes from the PLC, delete recipes, and provide some simple security for these functions. Adding those capabilities can all be done using existing Visual Basic technologies. Reading from the PLC may be done the same way it was in the simple read examples except that a `.Trigger` method would be used instead of `.Autopoll` to execute a one-time read of the recipe data from the PLC.

DATA LOGGING AND REAL-TIME TRENDING

This example reads two variables from the PLC and plots them on a trend chart while logging them to a database at the same time. The connection to the PLC is accomplished as before using the PLC ActiveX control. In this example, the properties are set to read registers 1 through 4, of which 1 and 4 will be plotted and logged. The command buttons on the form are set to stop/start polling and logging at a 100 ms interval. The adding of new data values and logging the values to the database is done in the `PLCMachine1_OnReadDone()` event so that data is logged automatically every time the driver gets new data.

To accomplish the database connection, a data control is used like in the recipe example. This time the control is given the name `dbMachine1Data`, the database name is `datalogger.mdb`, and the record source is the `tblMachine1Data` table in the database. These settings are all made on the properties page for the data control. Only five lines of code are needed to log variables to the database. These lines are inserted into the `PLCMachine1_OnReadDone()` event and are shown in listing 5. The data control automatically manages the opening and closing of the database connection for the user.

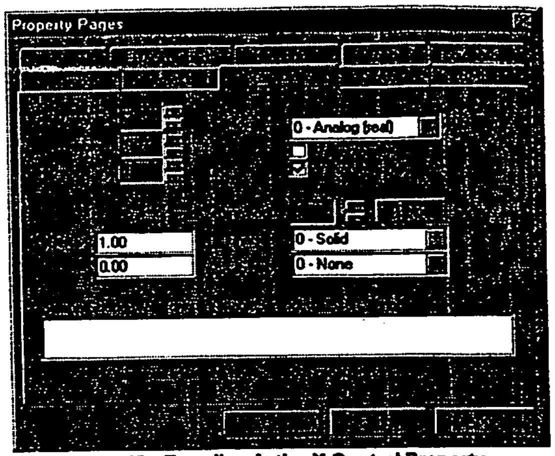
To trend the same data points on a real-time trend, another off-the-shelf ActiveX control is used. This control is placed on the form and given the name `Strip1`. To setup the control, several properties are set to facilitate the display of more than one pen and the pen colors. Also, some simple code is executed at runtime in the `Form1_Load` event to set the x-axis of the chart for the current time and the span of the chart to 30 seconds. Most of the properties of the control are left at their default values. The remainder are set using a tabbed dialog box shown in Figure 12.

Listing 5 - Code to add a datapoint record to the database

```
'add a record to the table
 dbMachine1Data.Recordset.AddNew
'set the Time field equal to the current time
 dbMachine1Data.Recordset("Time") = Now
'set the Variable 1 field to the first value read from the PLC
 dbMachine1Data.Recordset("Variable1") = PLCMachine1.WordVal(0)
'set the Variable 2 field to the 4th value read from the PLC
 dbMachine1Data.Recordset("Variable2") = PLCMachine1.WordVal(3)
'update the record in the database on disk
 dbMachine1Data.Recordset.Update
```

Listing 6 - Update the trend chart object Strip1 with the latest datapoints from the PLC

```
Strip1.AddXY 0, Now, PLCMachine1.WordVal(0)
Strip1.AddXY 1, Now, PLCMachine1.WordVal(3)
```


Property Pages

0 - Analog (real)

1.00

0.00

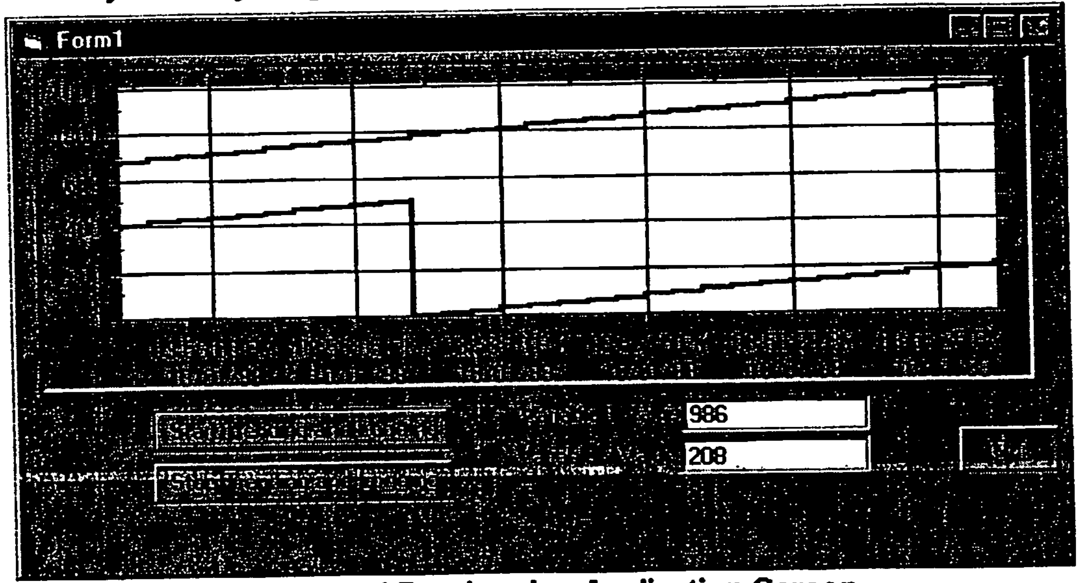
0 - Solid

0 - None

Figure 12 - Trending ActiveX Control Property Configuration Form with Tabbed Dialog Box

The code added to the PLCMachine1_OnReadDone() event is shown in Listing 6. These two lines of code add a new value to the chart each time the PLC ActiveX control reads another time point from the PLC. The timing of this plotting is driven by the read rate set with the AutoPoll method invoked on the PLCMachine1 control when the user clicks on the command button to start polling.

The resulting display is shown in Figure 13. For display of historical data, the strip chart control supports data binding with a data control just like the text boxes used in the recipe example. Using this capability, the chart may be easily setup to replay historical data from the database.


Form1

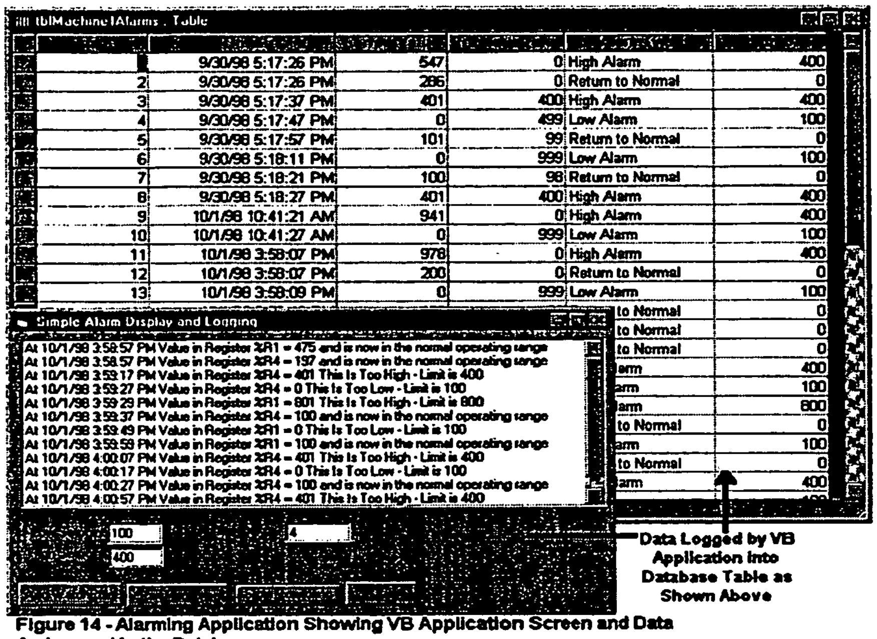
986

208

Figure 13 - Trending and Datalogging Application Screen

ALARMING

To accomplish the monitoring of analog values for alarm generation, a special method available on the PLC communications ActiveX control is utilized. This set of methods are referred to as watchpoints. The example screen shown in Figure 14 was built like all previous ones using simple Visual Basic controls and adding an instance of the PLC communications ActiveX control and setting it up as before with the name `PLCMachine1`. An instance of the data control named `dbMachine1Alarms` was added to connect to the table `tblMachine1Alarms` in the database `alarms.mdb` for the logging of alarm data. This example will use the same concepts used for logging the trend data for logging alarm data. When alarm must be logged, the `.Addnew` method is called on the database control's record set and the fields in the new record are populated with the alarm data, and the `.UpdateRecord` method is invoked to update the database.


tblMachine1Alarms : Table
Simple Alarm Display and Logging
Data Logged by VB Application Into Database Table as Shown Above

1	9/30/98 5:17:26 PM	547	0	High Alarm	400
2	9/30/98 5:17:26 PM	286	0	Return to Normal	0
3	9/30/98 5:17:37 PM	401	400	High Alarm	400
4	9/30/98 5:17:47 PM	0	499	Low Alarm	100
5	9/30/98 5:17:57 PM	101	99	Return to Normal	0
6	9/30/98 5:18:11 PM	0	999	Low Alarm	100
7	9/30/98 5:18:21 PM	100	98	Return to Normal	0
8	9/30/98 5:18:27 PM	401	400	High Alarm	400
9	10/1/98 10:41:21 AM	941	0	High Alarm	400
10	10/1/98 10:41:27 AM	0	999	Low Alarm	100
11	10/1/98 3:58:07 PM	978	0	High Alarm	400
12	10/1/98 3:58:07 PM	200	0	Return to Normal	0
13	10/1/98 3:58:09 PM	0	999	Low Alarm	100
				to Normal	0
				to Normal	0
				to Normal	0
				arm	400
				arm	100
				arm	800
				to Normal	0
				arm	100
				to Normal	0
				arm	400

```
At 10/1/98 3:58:57 PM Value in Register %R1 = 475 and is now in the normal operating range
At 10/1/98 3:58:57 PM Value in Register %R4 = 197 and is now in the normal operating range
At 10/1/98 3:59:17 PM Value in Register %R4 = 401 This Is Too High - Limit is 400
At 10/1/98 3:59:27 PM Value in Register %R4 = 0 This Is Too Low - Limit is 100
At 10/1/98 3:59:29 PM Value in Register %R1 = 801 This Is Too High - Limit is 800
At 10/1/98 3:59:37 PM Value in Register %R4 = 100 and is now in the normal operating range
At 10/1/98 3:59:49 PM Value in Register %R1 = 0 This Is Too Low - Limit is 100
At 10/1/98 3:59:59 PM Value in Register %R1 = 100 and is now in the normal operating range
At 10/1/98 4:00:07 PM Value in Register %R4 = 401 This Is Too High - Limit is 400
At 10/1/98 4:00:17 PM Value in Register %R4 = 0 This Is Too Low - Limit is 100
At 10/1/98 4:00:27 PM Value in Register %R4 = 100 and is now in the normal operating range
At 10/1/98 4:00:57 PM Value in Register %R4 = 401 This Is Too High - Limit is 400
```

100 | 4 | 400

Figure 14 - Alarming Application Showing VB Application Screen and Data As Logged In the Database

The major new concept utilized in this example is the generation of the actual alarm. The ActiveX control utilized for PLC communications has a set of methods that can be set to monitor for high, low, return to normal, or any change of data points specified by the developer. For simplicity, the PLC ActiveX control in this example is setup to read 10 data points in a block from the PLC by setting the `.Size` property. The user interface shown in Figure 14 allows the user to pick the register address and the high/low limits for each point. When the command button `cmdAddWatchpoint` is clicked, the code shown in listing 7 is executed to add the point to the internal list of "watchpoints" that the ActiveX control manages. After the user has setup all the desired watch points, they click on the button `cmdStartMonitoring`, which invokes the `.Autopoll` method on the `PLCMachine1` control to begin monitoring the datapoints at rate of every 100ms. The `cmdStopMonitoring` button is supplied purely for demonstration purposes. In a full fledged application, the starting of polling could be made automatic to prevent unauthorized turning off of alarming capabilities.

When the `PLCMachine1` control detects that any of the set alarm conditions are met, it generates an event. For example, for a high alarm event, the `PLCMachine1_OnAnalogHigh` event is generated. The event passes back variables containing the index to the point generating the alarm, the current value, the previous value, and the high limit level. In the alarm handler code shown in listing 8, a message is formatted and displayed on the list box and the code necessary to log the alarm to the database is executed. Similar code executes for low alarm and return to normal alarms.

By utilizing the watch points capability in the PLC communications ActiveX control, the developer does not have to manage making the comparisons for alarm generation. The developer focuses on providing a user interface for setting up alarm conditions and displaying the alarm and determining what data needs to be logged and in what format. Built-in tools in Visual Basic provide for the data logging to any database that may be connected to using ODBC and OLE-DB technologies.

Listing 7 - Code to add a watchpoint setup to the PLC ActiveX driver and notify the user that it has been successfully added by displaying a message box. The second subroutine is executed when the user clicks on the command button labeled "Start Monitoring".

```
Private Sub cmdAddWatchpoint_Click()
    intHighlimit = Int(txtHighLimit)   'get the high limit and convert to integer for this example
    intLowLimit = Int(txtLowLimit)   'do the same for the low limit
    intAddress = Int(txtAddress) - 1  'do the same and offset the address to 0 based addressing
    '
    ' Set the watch point in the PLCMachine1 ActiveX control by calling the WatchpointAdd
    ' method
    '
    PLCMachine1.WatchPointAdd vbInteger, intAddress, 0, 7, intLowLimit, intHighlimit, True
    '
    ' Display a message box advising the user the watchpoint has been added
    '
    result = MsgBox("Watchpoint Added for %R" & (intAddress + 1) & ", Low Limit=" & intLowLimit & ", High Limit=" &
    intHighlimit, vbOKOnly, "Watchpoint Added Successfully")
End Sub

Private Sub cmdStartMonitoring_Click()
    lstDisplayAlarms.Clear           'clear the alarm display - this is totally optional!
    PLCMachine1.AutoPoll (100)  'Start monitoring the watched points at 100ms rate
End Sub
```

Listing 8 - Code to display any high alarm that happens and log it to the database via the data control instance named dbMachine1Alarms.

```
Private Sub PLCMachine1_OnAnalogHigh(ByVal PtType As Integer, ByVal PtIndex As Integer, ByVal CurrentVal As Variant,
ByVal PreviousVal As Variant, ByVal HighLimit As Variant)
    '
    ' Build the display string and display it on the list box lstDisplayAlarms
    '
    txtHighAlarmMessage = "At " & Now & " Value in Register %R" & PtIndex + 1 & " = " & CurrentVal & " This Is Too High -
    Limit is " & HighLimit
    lstDisplayAlarms.AddItem txtHighAlarmMessage
    '
    ' Log the alarm data to the database
    '
    dbMachine1Alarms.Recordset.AddNew
    dbMachine1Alarms.Recordset("Time") = Now
    dbMachine1Alarms.Recordset("Currentvalue") = CurrentVal
    dbMachine1Alarms.Recordset("Previousvalue") = PreviousVal
    dbMachine1Alarms.Recordset("AlarmType") = "High Alarm"
    dbMachine1Alarms.Recordset("Limit") = HighLimit
    dbMachine1Alarms.UpdateRecord
End Sub
```

CONCLUSIONS

By utilizing Visual Basic's built in functionality for user interface design and database connectivity and adding off-the-shelf plug-in objects or ActiveX controls for graphics, trending, PLC communications, and alarming, developers, OEMs, integrators, and users can create industrial HMI applications. These applications have the flexibility to be designed in any way the user desires and are highly cost effective because they may be distributed in compiled form to many machines with few or no per machine royalty fees. The open architecture of Visual Basic allowing for the use of COM-based or ActiveX control plug-ins insures the developer will have choices when picking tools for building HMI applications in Visual Basic. The presence of over 3,000,000 people worldwide who know Visual Basic provides investment protection for future modification and support of the application.